

A Rudimentary Quantum Compiler

Robert R. Tucci
P.O. Box 226
Bedford, MA 01730
tucci@ar-tiste.com

February 1, 2008

Abstract

We present a new algorithm for reducing an arbitrary unitary matrix into a sequence of elementary operations (operations such as controlled-nots and qubit rotations). Such a sequence of operations can be used to manipulate an array of quantum bits (i.e., a quantum computer). We report on a C++ program called “Qubiter” that implements our algorithm. Qubiter source code is publicly available.

1. Introduction

1(a) Previous Work

In classical computation and digital electronics, one deals with sequences of elementary operations (operations such as AND, OR and NOT). These sequences are used to manipulate an array of classical bits. The operations are elementary in the sense that they act on only a few bits (1 or 2) at a time. Henceforth, we will sometimes refer to sequences as products and to operations as operators, instructions, steps or gates. Furthermore, we will abbreviate the phrase “sequence of elementary operations” by “SEO”. In quantum computation[1], one also deals with SEOs (with operations such as controlled-nots and qubit rotations), but for manipulating quantum bits (qubits) instead of classical bits. SEOs are often represented graphically by a qubit circuit.

In quantum computation, one often knows the unitary operator U that describes the evolution of an array of qubits. One must then find a way to reduce U into a SEO. In this paper, we present a new algorithm for accomplishing this task. We also report on a C++ program called “Qubiter” that implements our algorithm. Qubiter source code is publicly available at www.ar-tiste.com/qubiter.html. We call Qubiter a “quantum compiler” because, like a classical compiler, it produces a SEO for manipulating bits.

Our algorithm is general in the sense that it can be applied to any unitary operator U . Certain U are known to be polynomial in N_B ; i.e, they can be expressed as a SEO whose number of gates varies as a polynomial in N_B as N_B varies, where N_B is the number of bits. For example, a Discrete Fourier Transform (DFT) can be expressed as $\text{Order}(N_B^2)$ steps [2]. Our algorithm expresses a DFT as $\text{Order}(4^{N_B})$ steps. Hence, our algorithm is not “polynomially efficient”; i.e., it does not give a polynomial number of steps for all U for which this is possible. However, we believe that it is possible to introduce optimizations into the algorithm so as to make it polynomially efficient. Future papers will report our progress in finding such optimizations.

Previous workers [3] have described another algorithm for reducing a unitary operator into a SEO. Like ours, their algorithm is general, and it is not expected to be polynomial efficient without optimizations. Our algorithm is significantly different from theirs. Theirs is based on a mathematical technique described in Refs.[4]-[5], whereas ours is based on a mathematical technique called CS Decomposition[6]-[8], to be described later.

Quantum Bayesian (QB) Nets[9]-[10] are a method of modelling quantum systems graphically in terms of network diagrams. In a companion paper[11], we show how to apply the results of this paper to QB nets.

1(b) CS Decomposition

As mentioned earlier, our algorithm utilizes a mathematical technique called CS Decomposition[6]-[8]. In this name, the letters C and S stand for “cosine” and “sine”. Next we will state the special case of the CS Decomposition Theorem that arises in our algorithm.

Suppose that U is an $N \times N$ unitary matrix, where N is an even number. Then the CS Decomposition Theorem states that one can always express U in the form

$$U = \begin{bmatrix} L_0 & 0 \\ 0 & L_1 \end{bmatrix} D \begin{bmatrix} R_0 & 0 \\ 0 & R_1 \end{bmatrix}, \quad (1.1)$$

where L_0, L_1, R_0, R_1 are $\frac{N}{2} \times \frac{N}{2}$ unitary matrices and

$$D = \begin{bmatrix} D_{00} & D_{01} \\ D_{10} & D_{11} \end{bmatrix}, \quad (1.2a)$$

$$D_{00} = D_{11} = \text{diag}(C_1, C_2, \dots, C_{\frac{N}{2}}), \quad (1.2b)$$

$$D_{01} = \text{diag}(S_1, S_2, \dots, S_{\frac{N}{2}}), \quad (1.2c)$$

$$D_{10} = -D_{01}. \quad (1.2d)$$

For $i \in \{1, 2, \dots, \frac{N}{2}\}$, C_i and S_i are real numbers that satisfy

$$C_i^2 + S_i^2 = 1. \quad (1.2e)$$

Henceforth, we will use the term *D matrix* to refer to any matrix that satisfies Eqs.(1.2). If one partitions U into four blocks U_{ij} of size $\frac{N}{2} \times \frac{N}{2}$, then

$$U_{ij} = L_i D_{ij} R_j, \quad (1.3)$$

for $i, j \in \{0, 1\}$. Thus, D_{ij} gives the singular values of U_{ij} .

More general versions of the CS Decomposition Theorem allow for the possibility that we partition U into 4 blocks of unequal size.

Note that if U were a general (not necessarily unitary) matrix, then the four blocks U_{ij} would be unrelated. Then to find the singular values of the four blocks U_{ij} would require eight unitary matrices (two for each block), instead of the four L_i, R_j . This double use of the L_i, R_j is a key property of the CS decomposition.

1(c) Bird's Eye View of Algorithm

Our algorithm is described in detail in subsequent sections. Here we will only give a bird's eye view of it.

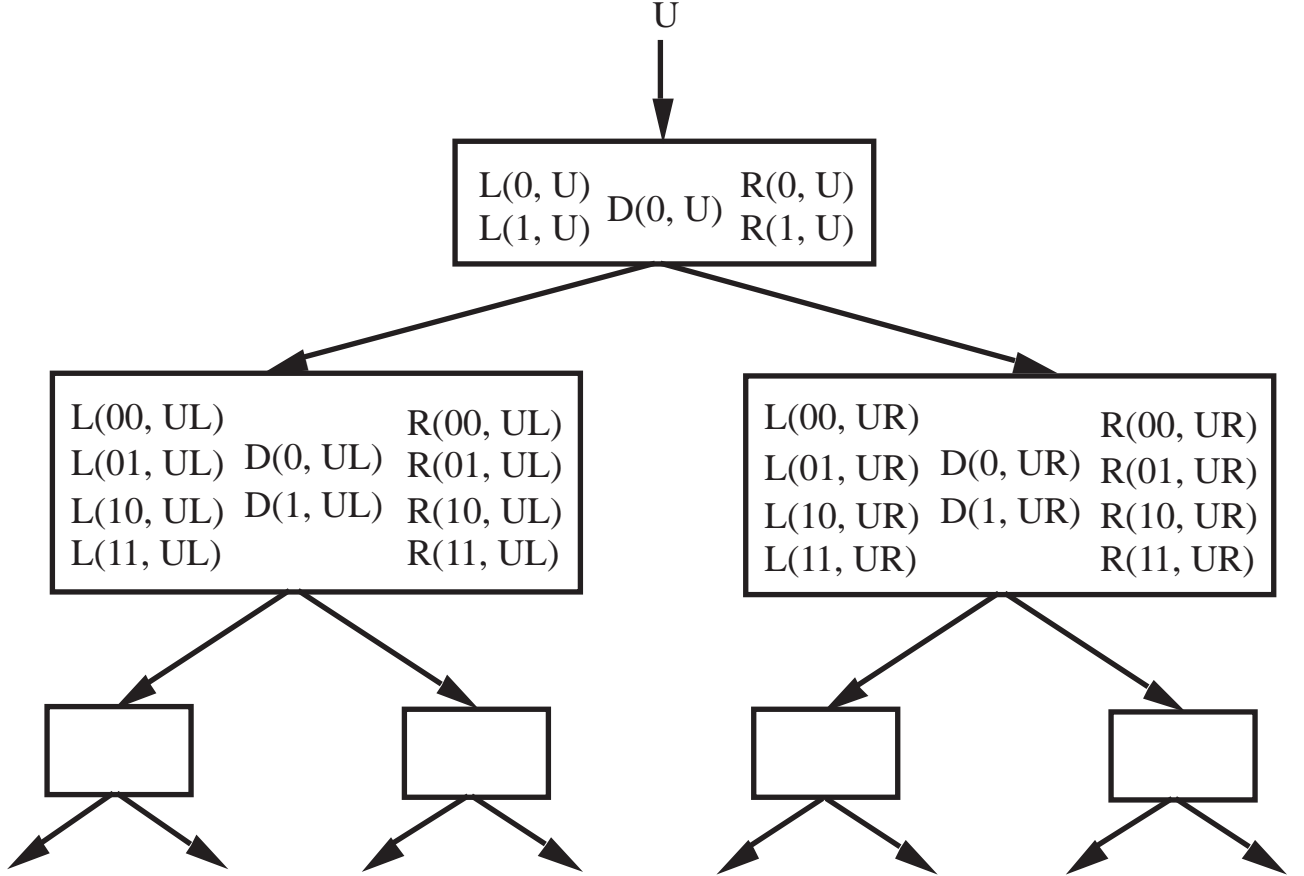


Fig.1 A binary tree. Each node β has a single parent. If the parent is to β 's right (ditto, left), then β contains the names of the matrices produced by applying the CS Decomposition Theorem to the L matrices (ditto, R matrices) of β 's parent.

Consider Fig.1. We start with a unitary matrix U . Without loss of generality, we can assume that the dimension of U is 2^{N_B} for some $N_B \geq 1$. (If initially U 's dimension is not a power of 2, we replace it by a direct sum $U \oplus I_r$ whose dimension is a power of two.) We apply the CS Decomposition method to U . This yields a matrix $D(0, U)$ of singular values, two unitary matrices $L(0, U)$ and $L(1, U)$ on the left and two unitary matrices $R(0, U)$ and $R(1, U)$ on the right. Then we apply the CS Decomposition method to each of the 4 matrices $L(0, U)$, $L(1, U)$, $R(0, U)$ and $R(1, U)$ that were produced in the previous step. Then we apply the CS Decomposition method to each of the 16 R and L matrices that were produced in the previous step. And so on. The lowest row of the pyramid in Fig.1 has L 's and R 's that are 1×1 dimensional, i.e., just complex numbers.

Call a *central matrix* either (1) a single D matrix, or (2) a direct sum $D_1 \oplus D_2 \oplus \dots \oplus D_r$ of D matrices, or (3) a diagonal unitary matrix. From Fig.1 it is clear that the initial matrix U can be expressed as a product of central matrices, with each node of the tree providing one of the central matrices in the product. Later on we will present techniques for decomposing any central matrix into a SEO.

2. Preliminaries

In this section, we introduce some notation and some general mathematical concepts that will be used in subsequent sections.

2(a) General Notation

We define $Z_{a,b} = \{a, a+1, \dots, b\}$ for any integers a and b . $\delta(x, y)$ equals one if $x = y$ and zero otherwise.

We will use the symbol N_B for the number (≥ 1) of bits and $N_S = 2^{N_B}$ for the number of states with N_B bits. Let $Bool = \{0, 1\}$. We will use lower case Latin letters $a, b, c \dots \in Bool$ to represent bit values and lower case Greek letters $\alpha, \beta, \gamma, \dots \in Z_{0, N_B-1}$ to represent bit positions. A vector such as $\vec{a} = a_{N_B-1} \dots a_2 a_1 a_0$ will represent a string of bit values, a_μ being the value of the μ 'th bit for $\mu \in Z_{0, N_B-1}$. A bit string \vec{a} has a decimal representation $d(\vec{a}) = \sum_{\mu=0}^{N_B-1} 2^\mu a_\mu$. For $\beta \in Z_{0, N_B-1}$, we will use $\vec{u}(\beta)$ to denote the β 'th standard unit vector, i.e, the vector with bit value of 1 at bit position β and bit value of zero at all other bit positions.

I_r will represent the r dimensional unit matrix. Suppose $\beta \in Z_{0, N_B-1}$ and M is any 2×2 matrix. We define $M(\beta)$ by

$$M(\beta) = I_2 \otimes \dots \otimes I_2 \otimes M \otimes I_2 \otimes \dots \otimes I_2, \quad (2.1)$$

where the matrix M on the right side is located at bit position β in the tensor product of N_B 2×2 matrices. The numbers that label bit positions in the tensor product increase from right to left (\leftarrow), and the rightmost bit is taken to be at position 0.

For any two square matrices A and B of the same dimension, we define the \odot product by $A \odot B = ABA^\dagger$, where A^\dagger is the Hermitian conjugate of A .

$\vec{\sigma} = (\sigma_x, \sigma_y, \sigma_z)$ will represent the vector of Pauli matrices, where

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \quad (2.2)$$

2(b) Sylvester-Hadamard Matrices

The Sylvester-Hadamard matrices[12] H_r are defined by:

$$H_1 = \begin{array}{c|cc} & 0 & 1 \\ \hline 0 & 1 & 1 \\ 1 & 1 & -1 \end{array}, \quad (2.3a)$$

$$H_2 = \begin{array}{c|cccc} & 00 & 01 & 10 & 11 \\ \hline 00 & 1 & 1 & 1 & 1 \\ 01 & 1 & -1 & 1 & -1 \\ 10 & 1 & 1 & -1 & -1 \\ 11 & 1 & -1 & -1 & 1 \end{array}, \quad (2.3b)$$

$$H_{r+1} = H_1 \otimes H_r, \quad (2.3c)$$

for any integer $r \geq 1$. In Eqs.(2.3), we have labelled the rows and columns with binary numbers in increasing order, and \otimes indicates a tensor product of matrices. From Eqs.(2.3), one can show that the entry of H_r at row \vec{a} and column \vec{b} is given by

$$(H_r)_{\vec{a}, \vec{b}} = (-1)^{\vec{a} \cdot \vec{b}}, \quad (2.4)$$

where $\vec{a} \cdot \vec{b} = \sum_{\mu=0}^{r-1} a_{\mu} b_{\mu}$. It is easy to check that

$$H_r^T = H_r, \quad (2.5)$$

$$H_r^2 = 2^r I_r. \quad (2.6)$$

In other words, H_r is a symmetric matrix, and the inverse of H_r equals H_r divided by however many rows it has.

2(c) Permutations

Subsequent sections will use the following very basic facts about permutations. For more details, see, for example, Ref.[13].

A *permutation* is a 1-1 onto map from a finite set X onto itself. The set of permutations on set X is a group if group multiplication is taken to be function composition. S_n , the *symmetric group in n letters*, is defined as the group of all permutations on any set X with n elements. If $X = Z_{1,n}$, then a permutation G which maps $i \in X$ to $a_i \in X$ (where $i \neq j$ implies $a_i \neq a_j$) can be represented by a matrix with entries

$$(G)_{j,i} = \delta(a_j, i), \quad (2.7)$$

for all $i, j \in X$. Note that all entries in any given row or column equal zero except for one entry which equals one. An alternative notation for G is

$$G = \begin{pmatrix} 1 & 2 & 3 & \cdots & n \\ a_1 & a_2 & a_3 & \cdots & a_n \end{pmatrix}. \quad (2.8)$$

The product of two symbols of the type shown in Eq.(2.8) is defined by function composition. For example,

$$\begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 \\ a_1 & a_2 & a_3 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ b_1 & b_2 & b_3 \end{pmatrix}. \quad (2.9)$$

Note how we have applied the permutations on the left side from right to left (\leftarrow). (Careful: Some authors apply them in the opposite direction (\rightarrow)). A cycle is a special type of permutation. If $G \in S_n$ maps $a_1 \rightarrow a_2$, $a_2 \rightarrow a_3$, \dots , $a_{r-1} \rightarrow a_r$, $a_r \rightarrow a_1$, where $i \neq j$ implies $a_i \neq a_j$ and $r \leq n$, then we call G a *cycle*. G may be represented as in Eqs.(2.7) and (2.8). Another way to represent it is by

$$G = (a_1, a_2, a_3, \dots, a_r). \quad (2.10)$$

(Careful: some people write $(a_r, \dots, a_3, a_2, a_1)$ instead.) We say that the cycle of Eq.(2.10) has *length* r . Cycles of length 1 are just the identity map. A cycle of length 2 is called a *transposition*. The product of two cycles need not be another cycle. For example,

$$(2, 1, 5)(1, 4, 5, 6) = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 4 & 1 & 3 & 2 & 6 & 5 \end{pmatrix} \quad (2.11)$$

cannot be expressed as a single cycle. Any permutation can be written as a product of cycles. For example,

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 4 & 1 & 3 & 2 & 6 & 5 \end{pmatrix} = (5, 6)(1, 4, 2). \quad (2.12)$$

The cycles on the right side of Eq.(2.12) are *disjoint*; i.e., they have no elements in common. Disjoint cycles commute. Any cycle can be expressed as a product of transpositions (assuming a group with ≥ 2 elements), by using identities such as:

$$(a_1, a_2, \dots, a_n) = (a_1, a_2)(a_2, a_3) \cdots (a_{n-1}, a_n), \quad (2.13)$$

$$(a_1, a_2, \dots, a_n) = (a_1, a_n) \cdots (a_1, a_3)(a_1, a_2). \quad (2.14)$$

Another useful identity is

$$(a, b) = (a, p)(p, b)(a, p). \quad (2.15)$$

This last identity can be applied repeatedly. For example, applied twice, it gives

$$(a, b) = (a, p_1)(p_1, b)(a, p_1) = (a, p_1)(p_1, p_2)(p_2, b)(p_1, p_2)(a, p_1) . \quad (2.16)$$

Since any permutation equals a product of cycles, and each of those cycles can be expressed as a product of transpositions, all permutations can be expressed as a product of transpositions (assuming a group with ≥ 2 elements). The decomposition of a permutation into transpositions is not unique. However, the number of transpositions whose product equals a given permutation is always either even or odd. An *even* (ditto, *odd*) *permutation* is defined as one which equals an even (ditto, odd) number of transpositions.

2(d) Projection Operators

Consider a single qubit first.

The qubit's basis states $|0\rangle$ and $|1\rangle$ will be represented by

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} , \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} . \quad (2.17)$$

The number operator n of the qubit is defined by

$$n = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} = \frac{1 - \sigma_z}{2} . \quad (2.18)$$

Note that

$$n|0\rangle = 0 , \quad n|1\rangle = |1\rangle . \quad (2.19)$$

We will often use \bar{n} as shorthand for

$$\bar{n} = 1 - n = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} = \frac{1 + \sigma_z}{2} . \quad (2.20)$$

Define P_0 and P_1 by

$$P_0 = \bar{n} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} , \quad P_1 = n = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} . \quad (2.21)$$

P_0 and P_1 are orthogonal projectors and they add to one:

$$P_a P_b = \delta(a, b) P_b \quad \text{for } a, b \in \text{Bool} , \quad (2.22a)$$

$$P_0 + P_1 = I_2 . \quad (2.22b)$$

Now consider N_B bits instead of just one.

For $\beta \in Z_{0, N_B-1}$, we define $P_0(\beta)$, $P_1(\beta)$, $n(\beta)$ and $\bar{n}(\beta)$ by means of Eq.(2.1).[14]

For $\vec{a} \in \text{Bool}^{N_B}$, let

$$P_{\vec{a}} = P_{a_{N_B-1}} \otimes \cdots \otimes P_{a_2} \otimes P_{a_1} \otimes P_{a_0} . \quad (2.23)$$

Note that

$$\sum_{\vec{a} \in \text{Bool}^{N_B}} P_{\vec{a}} = I_2 \otimes I_2 \otimes \cdots \otimes I_2 = I_{2^{N_B}} . \quad (2.24)$$

For example, with 2 bits we have

$$P_{00} = P_0 \otimes P_0 = \text{diag}(1, 0, 0, 0) , \quad (2.25a)$$

$$P_{01} = P_0 \otimes P_1 = \text{diag}(0, 1, 0, 0) , \quad (2.25b)$$

$$P_{10} = P_1 \otimes P_0 = \text{diag}(0, 0, 1, 0) , \quad (2.25c)$$

$$P_{11} = P_1 \otimes P_1 = \text{diag}(0, 0, 0, 1) , \quad (2.25d)$$

$$\sum_{a,b \in \text{Bool}} P_{a,b} = I_4 . \quad (2.26)$$

For $r \geq 1$, suppose P_1, P_2, \dots, P_r are orthogonal projection operators (i.e., $P_i P_j = \delta(i, j) P_j$), and $\alpha_1, \alpha_2, \dots, \alpha_r$ are complex numbers. Then it is easy to show by Taylor expansion that

$$\exp\left(\sum_{i=1}^r \alpha_i P_i\right) = \sum_{i=1}^r \exp(\alpha_i) P_i + \left(1 - \sum_{i=1}^r P_i\right) . \quad (2.27)$$

In other words, one can “pull out” the sum over P_i ’s from the argument of the exponential, but only if one adds a compensating term $1 - \sum_i P_i$ so that both sides of the equation agree when all the α_i ’s are zero.

3. State Permutations that Act on Two Bits

The goal of this paper is to reduce unitary matrices into qubit rotations and controlled-nots (c-nots). A qubit rotation (i.e., $\exp[i\vec{\theta} \cdot \vec{\sigma}(\beta)]$ for $\beta \in Z_{0, N_B-1}$ and some real 3-dimensional vector $\vec{\theta}$) acts on a single qubit at a time. This section will discuss gates such as c-nots that are state permutations that act on two bits at a time.

3(a) $N_B = 2$

Consider first the case when there are only 2 bits. Then there are four possible states—00, 01, 10, 11. With these 4 states, one can build 6 distinct transpositions:

$$(00, 01) = \begin{bmatrix} \sigma_x & \\ & I_2 \end{bmatrix} = P_0 \otimes \sigma_x + P_1 \otimes I_2 = \sigma_x(0)^{\bar{n}(1)} , \quad (3.1a)$$

$$(00, 10) = \begin{bmatrix} P_1 & P_0 \\ P_0 & P_1 \end{bmatrix} = I_2 \otimes P_1 + \sigma_x \otimes P_0 = \sigma_x(1)^{\bar{n}(0)} , \quad (3.1b)$$

$$(00, 11) = \begin{bmatrix} & & 1 \\ & I_2 & \\ 1 & & \end{bmatrix} , \quad (3.1c)$$

$$(01, 10) = \begin{bmatrix} 1 & & \\ & \sigma_x & \\ & & 1 \end{bmatrix} , \quad (3.1d)$$

$$(01, 11) = \begin{bmatrix} P_0 & P_1 \\ P_1 & P_0 \end{bmatrix} = I_2 \otimes P_0 + \sigma_x \otimes P_1 = \sigma_x(1)^{n(0)} , \quad (3.1e)$$

$$(10, 11) = \begin{bmatrix} I_2 & \\ & \sigma_x \end{bmatrix} = P_0 \otimes I_2 + P_1 \otimes \sigma_x = \sigma_x(0)^{n(1)} , \quad (3.1f)$$

where matrix entries left blank should be interpreted as zero. The rows and columns of the above matrices are labelled by binary numbers in increasing order (as in Eq.(2.3b) for H_2). Note that the 4 transpositions Eqs.(3.1 a,b,e,f) change only one bit value, whereas the other 2 transpositions Eqs.(3.1 c,d) change both bit values. We will call (00, 11) the *Twin-to-twin-er* and (01, 10) the *Exchanger*.

Exchanger[15] has four possible representations as a product of c-nots:

$$(01, 10) = (01, 00)(00, 10)(01, 00) = \sigma_x(0)^{\bar{n}(1)}\sigma_x(1)^{\bar{n}(0)}\sigma_x(0)^{\bar{n}(1)} , \quad (3.2a)$$

$$(01, 10) = (10, 11)(11, 01)(10, 11) = \sigma_x(0)^{n(1)}\sigma_x(1)^{n(0)}\sigma_x(0)^{n(1)} , \quad (3.2b)$$

$$(01, 10) = (10, 00)(00, 01)(10, 00) = \sigma_x(1)^{\bar{n}(0)}\sigma_x(0)^{\bar{n}(1)}\sigma_x(1)^{\bar{n}(0)} , \quad (3.2c)$$

$$(01, 10) = (01, 11)(11, 10)(01, 11) = \sigma_x(1)^{n(0)}\sigma_x(0)^{n(1)}\sigma_x(1)^{n(0)} . \quad (3.2d)$$

Note that one can go from Eq.(3.2a) to (3.2b) by exchanging n and \bar{n} ; from Eq.(3.2a) to (3.2c) by exchanging bit positions 0 and 1; from Eq.(3.2a) to (3.2d) by doing both, exchanging n and \bar{n} and exchanging bit positions 0 and 1. We will often represent Exchanger by $E(0, 1)$. It is easy to show that

$$E^T(0, 1) = E(0, 1) = E^{-1}(0, 1) , \quad (3.3a)$$

$$E(0, 1) = E(1, 0) , \quad (3.3b)$$

$$E^2(0, 1) = 1 . \quad (3.3c)$$

Furthermore, if X and Y are any two arbitrary 2×2 matrices, then, by using the matrix representation Eq.(3.1d) of Exchanger, one can show that

$$E(1, 0) \odot (X \otimes Y) = Y \otimes X . \quad (3.4)$$

Thus, Exchanger exchanges the position of matrices X and Y in the tensor product.

Twin-to-twin-er also has 4 possible representations as a product of c-nots. One is

$$(00, 11) = (00, 01)(01, 11)(00, 01) = \sigma_x(0)^{\bar{n}(1)}\sigma_x(1)^{n(0)}\sigma_x(0)^{\bar{n}(1)} . \quad (3.5)$$

As with Exchanger, the other 3 representations are obtained by replacing (1) n and \bar{n} , (2) bit positions 0 and 1, (3) both.

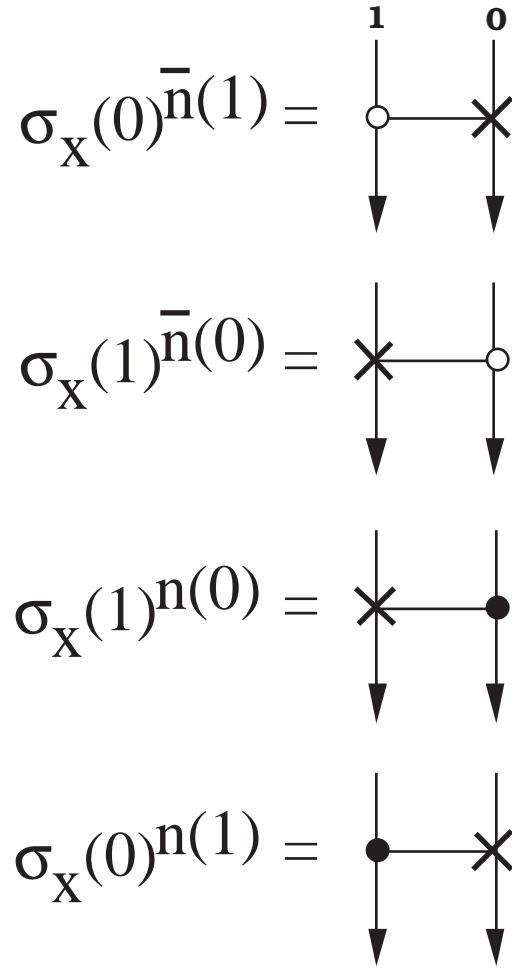


Fig.2 Circuit symbols for the 4 different types of c-nots.

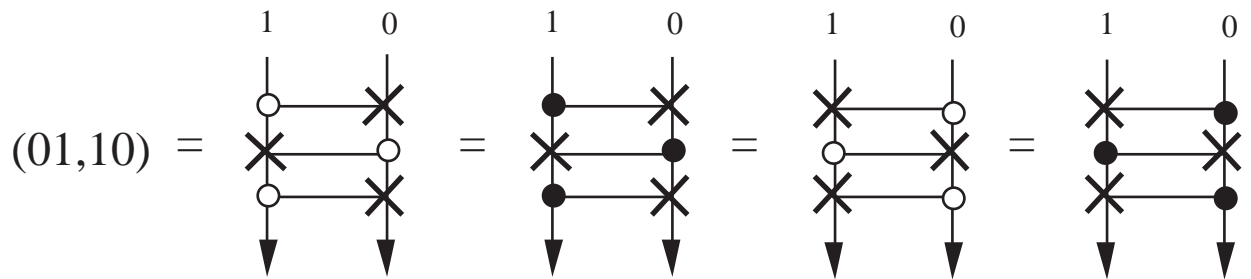


Fig.3 Four equivalent circuit diagrams for Exchanger.

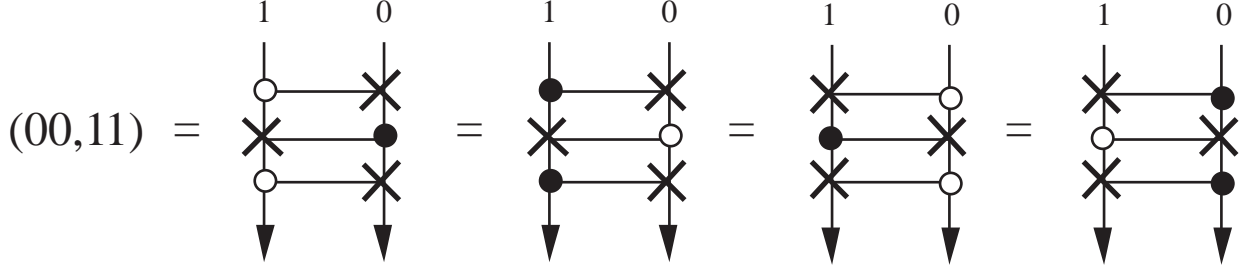


Fig.4 Four equivalent circuit diagrams for Twin-to-twin-er.

Figures 2, 3 and 4 give a diagrammatic representation of the 6 possible transpositions of states for $N_B = 2$.

3(b) Any $N_B \geq 2$

Suppose $a_1, b_1, a_2, b_2 \in Bool$ and $\alpha, \beta \in Z_{0, N_B-1}$. We define

$$(a_1 b_1, a_2 b_2)_{\alpha, \beta} = \prod_{(\Lambda, \Lambda', \Lambda'') \in Bool^{N_B-2}} (\Lambda a_1 \Lambda' b_1 \Lambda'', \Lambda a_2 \Lambda' b_2 \Lambda''), \quad (3.6)$$

where on the right side, a_1, a_2 are located at bit position α , and b_1, b_2 are located at bit position β . (Note that the transpositions on the right side of Eq.(3.6) are disjoint so they commute.) For example, for $N_B = 3$,

$$\sigma_x(0)^{n(1)} = (10, 11)_{1,0} = \prod_{a \in Bool} (a10, a11) = (010, 011)(110, 111). \quad (3.7)$$

Clearly, any permutation of states with N_B bits that permutes only 2 bits (i.e., Exchanger, Twin-to-twin-er, and all c-nots) can be represented by $(a_1 b_1, a_2 b_2)_{\alpha, \beta}$.

For $\alpha, \beta \in Z_{0, N_B-1}$, let $E(\alpha, \beta)$ represent Exchanger:

$$E(\alpha, \beta) = (01, 10)_{\alpha, \beta}. \quad (3.8)$$

As in the $N_B = 2$ case, $E(\alpha, \beta)$ can be expressed as a product of c-nots in 4 different ways. One way is

$$E(\alpha, \beta) = \sigma_x(\alpha)^{n(\beta)} \sigma_x(\beta)^{n(\alpha)} \sigma_x(\alpha)^{n(\beta)}. \quad (3.9)$$

The other 3 ways are obtained by exchanging (1) n and \bar{n} , (2) bit positions α and β , (3) both. Again as in the $N_B = 2$ case,

$$E^T(\alpha, \beta) = E(\alpha, \beta) = E^{-1}(\alpha, \beta), \quad (3.10a)$$

$$E(\alpha, \beta) = E(\beta, \alpha), \quad (3.10b)$$

$$E^2(\alpha, \beta) = 1 . \quad (3.10c)$$

Furthermore, if X and Y are two arbitrary 2×2 matrices and $\alpha, \beta \in Z_{0, N_B-1}$ such that $\alpha \neq \beta$, then

$$E(\alpha, \beta) \odot [X(\alpha)Y(\beta)] = X(\beta)Y(\alpha) . \quad (3.11)$$

Equation (3.11) is an extremely useful result. It says that $E(\alpha, \beta)$ is a transposition of bit positions. (Careful: this is not the same as a transposition of bit states.) Furthermore, the $E(\alpha, \beta)$ generate the group of $N_B!$ permutations of bit positions. (Careful: this is not the same as the group of $(2^{N_B})!$ permutations of states with N_B bits.)

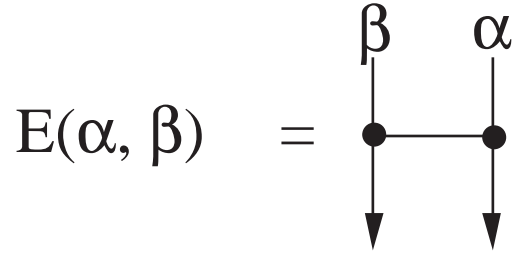


Fig.5 Circuit symbol for Exchanger.

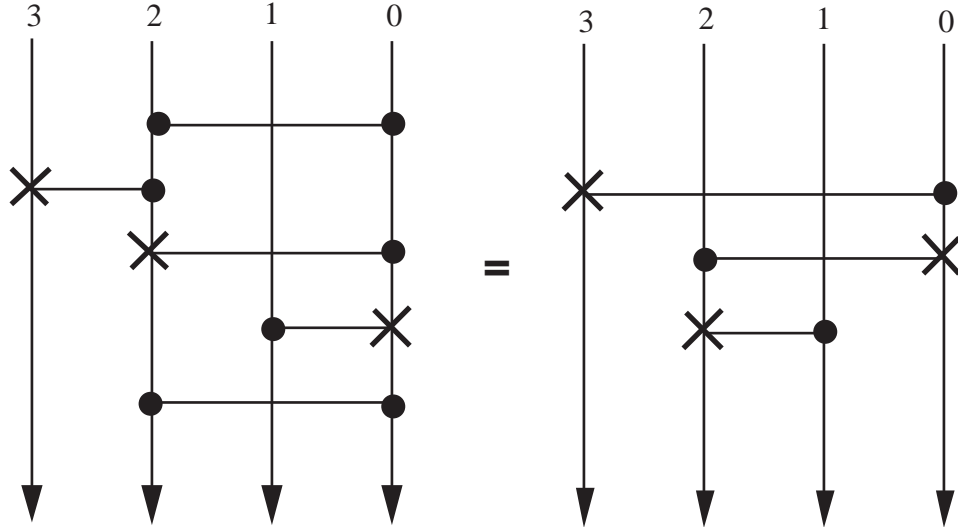


Fig.6 Circuit diagram for Eq.(3.12).

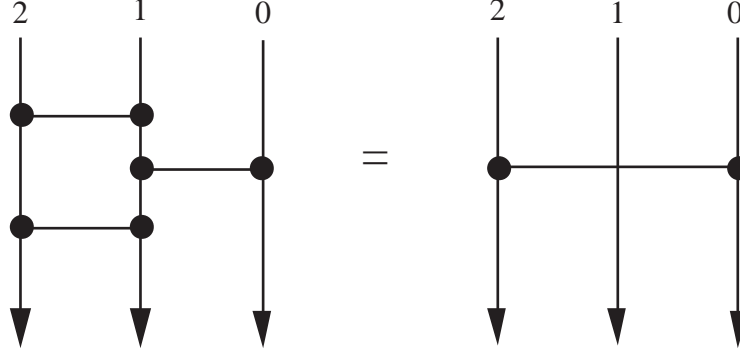


Fig.7 Circuit diagram for Eq.(3.14).

An example of how one can use Eq.(3.11) is

$$E(2, 0) \odot [\sigma_x(0)^{n(1)} \sigma_x(2)^{n(0)} \sigma_x(3)^{n(2)}] = \sigma_x(2)^{n(1)} \sigma_x(0)^{n(2)} \sigma_x(3)^{n(0)} . \quad (3.12)$$

Figure 5 gives a convenient way of representing $E(\alpha, \beta)$ diagrammatically. Using this symbol, the example of Eq.(3.12) can be represented by Fig.6.

Of course, identities that are true for a general transposition are also true for $E(\alpha, \beta)$. For example,

$$(2, 0) = (2, 1)(1, 0)(2, 1) . \quad (3.13)$$

Therefore,

$$E(2, 0) = E(2, 1)E(1, 0)E(2, 1) . \quad (3.14)$$

Figure 7 is a diagrammatic representation of Eq.(3.14).

4. Decomposing Central Matrix into SEO

In Section 1(d), we gave only a partial description of our algorithm. In this section, we complete that description by showing how to decompose each of the 3 possible kinds of central matrices into a SEO.

4(a)When Central Matrix is a Single D Matrix

D matrices are defined by Eqs.(1.2). They can be expressed in terms of projection operators as follows:

$$D = \sum_{\vec{a} \in \text{Bool}^{N_B-1}} \exp(i\phi_{\vec{a}} \sigma_y) \otimes P_{\vec{a}} , \quad (4.1)$$

where the $\phi_{\vec{a}}$ are real numbers. Note that in Eq.(4.1), \vec{a} has $N_B - 1$ components instead of the full N_B . Using the identity Eq.(2.27), one gets

$$D = \exp \left(i \sum_{\vec{a} \in \text{Bool}^{N_B-1}} \phi_{\vec{a}} \sigma_y \otimes P_{\vec{a}} \right). \quad (4.2)$$

Now define new angles $\theta_{\vec{b}}$ by

$$\phi_{\vec{a}} = \sum_{\vec{b} \in \text{Bool}^{N_B-1}} (-1)^{\vec{a} \cdot \vec{b}} \theta_{\vec{b}}. \quad (4.3)$$

Suppose $\vec{\phi}$ (ditto, $\vec{\theta}$) is a column vector whose components are the numbers $\phi_{\vec{a}}$ (ditto, $\theta_{\vec{a}}$) arranged in order of increasing \vec{a} . Then Eq.(4.3) is equivalent to

$$\vec{\phi} = H_{N_B-1} \vec{\theta}. \quad (4.4)$$

This is easily inverted to

$$\vec{\theta} = \frac{1}{2^{N_B-1}} H_{N_B-1} \vec{\phi}. \quad (4.5)$$

Let $A_{\vec{b}}$ for $\vec{b} \in \text{Bool}^{N_B-1}$ be defined by

$$A_{\vec{b}} = \exp \left(i \theta_{\vec{b}} \sigma_y \otimes \sum_{\vec{a} \in \text{Bool}^{N_B-1}} (-1)^{\vec{a} \cdot \vec{b}} P_{\vec{a}} \right). \quad (4.6)$$

Then D can be written as

$$D = \prod_{\vec{b} \in \text{Bool}^{N_B-1}} A_{\vec{b}}. \quad (4.7)$$

Note that the $A_{\vec{b}}$ operators on the right side commute so the order in which they are multiplied is irrelevant. Next we establish 2 useful identities: If $\beta \in Z_{0, N_B-1}$ and $\vec{u}(\beta) \in \text{Bool}^{N_B}$ is the β 'th standard unit vector, then

$$\begin{aligned} & \sum_{\vec{a} \in \text{Bool}^{N_B}} (-1)^{\vec{a} \cdot \vec{u}(\beta)} P_{\vec{a}} \\ &= I_2 \otimes \cdots \otimes I_2 \otimes \left[\sum_{a_\beta \in \text{Bool}} (-1)^{a_\beta} P_{a_\beta} \right] \otimes I_2 \otimes \cdots \otimes I_2 \\ &= P_0(\beta) - P_1(\beta) \\ &= \sigma_z(\beta) \end{aligned} \quad (4.8)$$

If $\beta, \alpha \in Z_{0, N_B-1}$ and $\alpha \neq \beta$, then

$$\begin{aligned} & \sigma_x(\beta)^{n(\alpha)} \odot \sigma_y(\beta) \\ &= [\sigma_x(\beta) P_1(\alpha) + P_0(\alpha)] \odot \sigma_y(\beta) \\ &= \sigma_y(\beta) [-P_1(\alpha) + P_0(\alpha)] \\ &= \sigma_y(\beta) \sigma_z(\alpha) \end{aligned} \quad (4.9)$$

Now we are ready to express $A_{\vec{b}}$ in terms of elementary operators. For any $\vec{b} \in \text{Bool}^{N_B-1}$, we can write

$$\vec{b} = \sum_{j=0}^{r-1} \vec{u}(\beta_j) , \quad (4.10)$$

where

$$N_B - 2 \geq \beta_{r-1} > \cdots > \beta_1 > \beta_0 \geq 0 . \quad (4.11)$$

In other words, \vec{b} has bit value of 1 at bit positions β_j . At all other bit positions, \vec{b} has bit value of 0. r is the number of bits in \vec{b} whose value is 1. When $\vec{b} = 0$, $r = 0$. Applying Eqs.(4.8) and (4.9), one gets

$$\begin{aligned} & [\sigma_x(N_B - 1)^{n(\beta_{r-1})} \cdots \sigma_x(N_B - 1)^{n(\beta_2)} \sigma_x(N_B - 1)^{n(\beta_1)}] \odot \sigma_y(N_B - 1) \\ &= \sigma_y(N_B - 1) \prod_{j=0}^{r-1} \sigma_z(\beta_j) \\ &= \prod_{j=0}^{r-1} \left(\sum_{\vec{a}} (-1)^{\vec{a} \cdot \vec{u}(\beta_j)} \sigma_y \otimes P_{\vec{a}} \right) \\ &= \sum_{\vec{a}} (-1)^{\vec{a} \cdot \vec{b}} \sigma_y \otimes P_{\vec{a}} \end{aligned} \quad (4.12)$$

Thus,

$$A_{\vec{b}} = [\sigma_x(N_B - 1)^{n(\beta_{r-1})} \cdots \sigma_x(N_B - 1)^{n(\beta_2)} \sigma_x(N_B - 1)^{n(\beta_1)}] \odot \exp[i\theta_{\vec{b}} \sigma_y(N_B - 1)] . \quad (4.13)$$

There are other ways of decomposing $A_{\vec{b}}$ into a SEO. For example, using the above method, one can also show that

$$A_{\vec{b}} = [\sigma_x(\beta_{r-2})^{n(\beta_{r-1})} \cdots \sigma_x(\beta_1)^{n(\beta_2)} \sigma_x(\beta_0)^{n(\beta_1)}] \odot \exp[i\theta_{\vec{b}} \sigma_y(\beta_0)] . \quad (4.14)$$

In conclusion, we have shown how to decompose a D matrix into a SEO. For example, suppose $N_B = 3$. Then

$$D = \sum_{a,b \in Bool} \exp(i\phi_{ab} \sigma_y) \otimes P_a \otimes P_b . \quad (4.15)$$

Define $\vec{\theta}$ by

$$\vec{\theta} = \frac{1}{4} H_2 \vec{\phi} . \quad (4.16)$$

By Eqs.(4.7) and (4.13),

$$D = A_{00} A_{01} A_{10} A_{11} , \quad (4.17)$$

where

$$A_{00} = \exp(i\theta_{00} \sigma_y) \otimes I_2 \otimes I_2 , \quad (4.18a)$$

$$A_{01} = \sigma_x(2)^{n(0)} \odot [\exp(i\theta_{01}\sigma_y) \otimes I_2 \otimes I_2] , \quad (4.18b)$$

$$A_{10} = \sigma_x(2)^{n(1)} \odot [\exp(i\theta_{10}\sigma_y) \otimes I_2 \otimes I_2] , \quad (4.18c)$$

$$A_{11} = [\sigma_x(2)^{n(1)}\sigma_x(2)^{n(0)}] \odot [\exp(i\theta_{11}\sigma_y) \otimes I_2 \otimes I_2] . \quad (4.18d)$$

4(b) When Central Matrix is a Direct Sum of D Matrices

Consider first the case $N_B = 3$. Let $R(\phi) = \exp(i\sigma_y\phi)$. Previously we used the fact that any D matrix D can be expressed as

$$D = \sum_{a,b \in Bool} R(\phi''_{ab}) \otimes P_a \otimes P_b . \quad (4.19)$$

But what if R were located at bit positions 0 or 1 instead of 2? By expressing both sides of the following equations as 8×8 matrices, one can show that

$$D_0 \oplus D_1 = \sum_{a,b \in Bool} P_a \otimes R(\phi'_{ab}) \otimes P_b , \quad (4.20)$$

$$D_{00} \oplus D_{01} \oplus D_{10} \oplus D_{11} = \sum_{a,b \in Bool} P_a \otimes P_b \otimes R(\phi_{ab}) , \quad (4.21)$$

where the D_j and D_{ij} are D matrices. One can apply a string of Exchangers to move R in Eqs.(4.20) and (4.21) to any bit position. Thus,

$$D_0 \oplus D_1 = E(1, 2) \odot \left(\sum_{a,b \in Bool} R(\phi'_{ab}) \otimes P_a \otimes P_b \right) , \quad (4.22)$$

$$D_{00} \oplus D_{01} \oplus D_{10} \oplus D_{11} = E(0, 1)E(1, 2) \odot \left(\sum_{a,b \in Bool} R(\phi_{ab}) \otimes P_a \otimes P_b \right) . \quad (4.23)$$

(Careful: $E(0, 2) \neq E(0, 1)E(1, 2)$. $E(0, 2)$ will change $\sum_{a,b} R(\phi_{ab}) \otimes P_a \otimes P_b$ to $\sum_{a,b} P_b \otimes P_a \otimes R(\phi_{ab})$, which is not the same as the left side of Eq.(4.21)).

For general $N_B \geq 1$, if $k \in Z_{0, N_B-1}$ and

$$E = \begin{cases} 1 & \text{if } k = 0 \text{ or } N_B = 1 \\ E(N_B - k - 1, N_B - k) \cdots E(N_B - 3, N_B - 2)E(N_B - 2, N_B - 1) & \text{otherwise} \end{cases} , \quad (4.24)$$

then a direct sum of 2^k D matrices can be expressed as

$$E \odot \left(\sum_{\vec{a} \in \text{Bool}^{N_B-1}} R(\phi_{\vec{a}}) \otimes P_{\vec{a}} \right) . \quad (4.25)$$

It follows that if we want to decompose a direct sum of D matrices into a SEO, we can do so in 2 steps: (1) decompose into a SEO the D matrix that one obtains by moving the qubit rotation to bit position $N_B - 1$, (2) Replace each bit name in the decomposition by its “alias”. By alias we mean the new name assigned by the bit permutation E defined by Eq.(4.24).

4(c)When Central Matrix is a Diagonal Unitary Matrix

Any diagonal unitary matrix C can be expressed as

$$C = \sum_{\vec{a} \in \text{Bool}^{N_B}} \exp(i\phi_{\vec{a}}) P_{\vec{a}} , \quad (4.26)$$

where the $\phi_{\vec{a}}$ are real numbers. Using the identity Eq.(2.27) yields

$$C = \exp \left(i \sum_{\vec{a} \in \text{Bool}^{N_B}} \phi_{\vec{a}} P_{\vec{a}} \right) . \quad (4.27)$$

Now define new angles $\theta_{\vec{b}}$ by

$$\phi_{\vec{a}} = \sum_{\vec{b} \in \text{Bool}^{N_B}} (-1)^{\vec{a} \cdot \vec{b}} \theta_{\vec{b}} . \quad (4.28)$$

In terms of vectors,

$$\vec{\phi} = H_{N_B} \vec{\theta} , \quad (4.29)$$

and

$$\vec{\theta} = \frac{1}{2^{N_B}} H_{N_B} \vec{\phi} . \quad (4.30)$$

Let $A_{\vec{b}}$ for $\vec{b} \in \text{Bool}^{N_B}$ be defined by

$$A_{\vec{b}} = \exp \left(i\theta_{\vec{b}} \sum_{\vec{a} \in \text{Bool}^{N_B}} (-1)^{\vec{a} \cdot \vec{b}} P_{\vec{a}} \right) . \quad (4.31)$$

Then C can be written as

$$C = \prod_{\vec{b} \in \text{Bool}^{N_B}} A_{\vec{b}} , \quad (4.32)$$

where the $A_{\vec{b}}$ operators commute. For any $\vec{b} \in \text{Bool}^{N_B}$, we can write

$$\vec{b} = \sum_{j=0}^{r-1} \vec{u}(\beta_j) , \quad (4.33)$$

where

$$N_B - 1 \geq \beta_{r-1} > \cdots > \beta_1 > \beta_0 \geq 0 . \quad (4.34)$$

(Careful: Compare this with Eq.(4.11). Now $\vec{b} \in Bool^{N_B}$ instead of $Bool^{N_B-1}$ and β_{r-1} can be as large as $N_B - 1$ instead of $N_B - 2$.) One can show using the techniques of Section 4(a) that

$$A_{\vec{b}} = \begin{cases} \exp[i\theta_0] & \text{if } r = 0 \\ \exp[i\theta_{\vec{b}}\sigma_z(\beta_0)] & \text{if } r = 1 \\ \left[\sigma_x(\beta_0)^{n(\beta_{r-1})} \cdots \sigma_x(\beta_0)^{n(\beta_2)} \sigma_x(\beta_0)^{n(\beta_1)} \right] \odot \exp[i\theta_{\vec{b}}\sigma_z(\beta_0)] & \text{if } r \geq 2 \end{cases} . \quad (4.35)$$

As in Section 4(a), there are other ways of decomposing $A_{\vec{b}}$ into a SEO.

In conclusion, we have shown how to decompose a diagonal unitary matrix into a SEO. For example, suppose $N_B = 2$. Then

$$C = \text{diag}(e^{i\phi_{00}}, e^{i\phi_{01}}, e^{i\phi_{10}}, e^{i\phi_{11}}) . \quad (4.36)$$

Define $\vec{\theta}$ by

$$\vec{\theta} = \frac{1}{4} H_2 \vec{\phi} . \quad (4.37)$$

By Eqs.(4.32) and (4.35),

$$C = A_{00} A_{01} A_{10} A_{11} , \quad (4.38)$$

where

$$A_{00} = \exp(i\theta_{00}) , \quad (4.39a)$$

$$A_{01} = I_2 \otimes \exp(i\theta_{01}\sigma_z) , \quad (4.39b)$$

$$A_{10} = \exp(i\theta_{10}\sigma_z) \otimes I_2 , \quad (4.39c)$$

$$A_{11} = \sigma_x(0)^{n(1)} \odot [I_2 \otimes \exp(i\theta_{11}\sigma_z)] . \quad (4.39d)$$

5. Qubiter

At present, Qubiter is a very rudimentary program. We hope that its fans will extend and enhance it in the future. Qubiter1.0 is written in pure C++, and has no graphical user interface. In its “compiling” mode, Qubiter takes as input a file with the entries of a unitary matrix and returns as output a file with a SEO. In its “decompiling” mode, it does the opposite: it takes a SEO file and returns the entries of a matrix. The lines in a SEO file are of 4 types:

(a) PHAS ang

where ang is a real number. This signifies a phase factor $\exp(i(ang)\frac{\pi}{180})$.

(b) CNOT α $char$ β

where $\alpha, \beta \in Z_{0, N_B-1}$ and $char \in \{T, F\}$. T and F stand for true and false. If $char$ is the character T , this signifies $\sigma_x(\beta)^{n(\alpha)}$. Read it as “c-not: if α is true, then flip β .” If $char$ is the character F , this signifies $\sigma_x(\beta)^{\bar{n}(\alpha)}$. Read it as “c-not: if α is false, then flip β .”

(c) ROTY α ang

where $\alpha \in Z_{0, N_B-1}$ and ang is a real number. This signifies the rotation of qubit α about the Y axis by an angle ang in degrees. In other words, $\exp(i\sigma_y(\alpha)ang\frac{\pi}{180})$.

(d) ROTZ α ang

This is the same as (c) except that the rotation is about the Z axis instead of the Y one.

As a example, consider what Qubiter gives for the Discrete Fourier Transform matrix U . This matrix has entries

$$U_{\vec{a}, \vec{b}} = \frac{1}{\sqrt{N_S}} \exp \left[\frac{i2\pi d(\vec{a})d(\vec{b})}{N_S} \right], \quad (5.1)$$

where $\vec{a}, \vec{b} \in Bool^{N_B}$. N_S and $d(\cdot)$ were defined in Section 2(a). When $N_B = 2$, Qubiter gives 33 operations (see Fig. 8). After doing the trivial optimization of removing all factors $A_{\vec{b}}$ for which the rotation angle is zero, the 33 operations in Fig.8 reduce to 25 operations in Fig.9. In the future, we plan to introduce into Qubiter many more optimizations and some quantum error correction. Much work remains to be done.

```

ROTZ 0 292.500000
ROTZ 1 45.0000000
CNOT 1 T 0
ROTZ 0 315.000000
CNOT 1 T 0
ROTY 0 315.000000
CNOT 1 T 0
ROTY 0 0.00000000
CNOT 1 T 0
ROTZ 0 337.500000
ROTZ 1 0.00000000
CNOT 1 T 0
ROTZ 0 45.0000000
CNOT 1 T 0
ROTY 1 315.000000
CNOT 0 T 1
ROTY 1 22.5000000
CNOT 0 T 1
ROTZ 0 247.500000
ROTZ 1 45.0000000
CNOT 1 T 0
ROTZ 0 315.000000
CNOT 1 T 0
ROTY 0 315.000000
CNOT 1 T 0
ROTY 0 0.00000000
CNOT 1 T 0
ROTZ 0 22.5000000
ROTZ 1 0.00000000
CNOT 1 T 0
ROTZ 0 315.000000
CNOT 1 T 0
PHAS 157.500000

```

Fig.8 Output of Qubiter with zero angle optimization turned OFF. (input: 2-bit Discrete Fourier Transform matrix).

```

ROTZ 0 292.500000
ROTZ 1 45.0000000
CNOT 1 T 0
ROTZ 0 315.000000
CNOT 1 T 0
ROTY 0 315.000000
ROTZ 0 337.500000
CNOT 1 T 0
ROTZ 0 45.0000000
CNOT 1 T 0
ROTY 1 315.000000
CNOT 0 T 1
ROTY 1 22.5000000
CNOT 0 T 1
ROTZ 0 247.500000
ROTZ 1 45.0000000
CNOT 1 T 0
ROTZ 0 315.000000
CNOT 1 T 0
ROTY 0 315.000000
ROTZ 0 22.5000000
CNOT 1 T 0
ROTZ 0 315.000000
CNOT 1 T 0
PHAS 157.500000

```

Fig.9 Output of Qubiter with zero angle optimization turned ON. (input: 2-bit Discrete Fourier Transform matrix).

References

- [1] D.P. DiVincenzo, Science **270**, 255 (1995).
- [2] D. Knuth, *The Art of Programming, Vol. 2, Seminumerical Algorithms* (Addison-Wesley, 1981); D. Coppersmith, IBM Research Rep. No. 19642, 1994; Robert B. Griffiths, Chi-Sheng Niu, Physical Review Letters **76**, 3228 (1994).
- [3] A. Barrenco, C.H. Bennett, R. Cleve, D.P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J.H. Smolin, H. Weinfurter, Physical Review A **52**, 3457 (1995).
- [4] F. D. Murnaghan, *The Orthogonal and Symplectic Groups* (Institute for Advanced Studies, Dublin, 1958).
- [5] M. Reck and A. Zeilinger, Physical Review Letters **73**, 58 (1994).
- [6] G.W. Stewart, Numerische Mathematik **40**, 297 (1982).
- [7] Charles Van Loan, Numerische Mathematik **46**, 479 (1985).
- [8] C.C. Paige, M. Wei, Linear Algebra And Its Applications **208**, 303 (1994).

- [9] R. R. Tucci, Int. Jour. of Mod. Physics **B9**, 295 (1995). Los Alamos eprint quant-ph/9706039
- [10] A commercial software program called “Quantum Fog” that simulates QB nets is available at www.ar-tiste.com .
- [11] R. R. Tucci, “How to Compile a Quantum Bayesian Net”. Los Alamos eprint quant-ph
- [12] R. K. Rao Yarlagadda, J. E. Hershey, *Hadamard Matrix Analysis and Synthesis* (Kluwer Academic Pub., 1997).
- [13] J. B. Fraleigh, *A First Course in Abstract Algebra* (Addison-Wesley, 1994).
- [14] Recently, these projection operators have been used very successfully to simplify the calculations dealing with NMR quantum computers. See S. S. Samaroo, D. G. Cory, T. F. Havel, Los Alamos eprint quant-ph/9801002.
- [15] The usefulness of Exchanger in quantum computation has been known for a long time: at least as early as R. P. Feynman’s paper “Quantum Mechanical Computers”, *Foundation of Physics* **16**, 507 (1986).

FIGURE CAPTIONS:

FIG.1 A binary tree. Each node β has a single parent. If the parent is to β 's right (ditto, left), then β contains the names of the matrices produced by applying the CS Decomposition Theorem to the L matrices (ditto, R matrices) of β 's parent.

FIG.2 Circuit symbols for the 4 different types of c-nots.

FIG.3 Four equivalent circuit diagrams for Exchanger.

FIG.4 Four equivalent circuit diagrams for Twin-to-twin-er.

FIG.5 Circuit symbol for Exchanger.

FIG.6 Circuit diagram for Eq.(3.12).

FIG.7 Circuit diagram for Eq.(3.14).

FIG.8 Output of Qubiter with zero angle optimization turned OFF. (input: 2-bit Discrete Fourier Transform matrix).

FIG.9 Output of Qubiter with zero angle optimization turned ON. (input: 2-bit Discrete Fourier Transform matrix).